Top 10 SQL Queries

Every Data Scientist Should Know

With Code, Tables, and Real-World Use Cases

Mostapha Kalami Heris

linkedin.com/in/smkalami

What You Will Learn

Practical SQL Patterns for Real-World Data Science

This guide is designed for **beginners and intermediate learners** who want to sharpen their SQL skills.

You will explore:

✅ How to query like a data scientist

✅ Interview-ready SQL examples

✅ Core use cases: filtering, joining, aggregating, ranking

✅ Bonus tips for real-world data challenges

> Learn by example. Practice by doing. Master by understanding.

Core SQL Skills for Data Scientists

The Building Blocks Behind All 10 Queries

⬇️ Filtering

```sql
SELECT * FROM users WHERE signup_date > '2025-01-01';
```

📊 Aggregation

```sql
SELECT country, COUNT(*) FROM users GROUP BY country;
```

🔗 Joining Tables

```sql
SELECT * FROM orders JOIN users ON orders.user_id = users.user_id;
```

🪟 Window Functions

```sql
RANK() OVER (ORDER BY total_spent DESC)
```

🧩 Subqueries & CTEs

```sql
WITH top_users AS (...) SELECT * FROM top_users;
```

> These tools let you write smarter, more powerful SQL.

Query #1: Count Unique Users Per Country

Skill: 📊 Aggregation

Scenario: You want to report how many distinct users are active in each country.

💻 SQL

```sql
SELECT country, COUNT(DISTINCT user_id) AS user_count
FROM users
GROUP BY country;
```

🧾 Input Table: `users`

user_id	name	country
1	Alice	USA
2	Bob	Canada
3	Carlos	USA
4	Diana	USA
5	Emma	Canada
6	Fiona	Germany

📊 Output

country	user_count
USA	3
Canada	2
Germany	1

✅ Why It Matters

- Learn to count distinct values by group
- A must-have skill for summary reports

> 💡 **Tip:** Always check for duplicates before using `COUNT(*)`.

Query #2: Filter Users Based on Recent Activity

Skill: ▼ Filtering

Scenario: You need a list of users who signed up in the last 30 days.

💻 SQL

```sql
SELECT user_id, name, signup_date
FROM users
WHERE signup_date >= CURRENT_DATE - INTERVAL '30 days';
```

🧾 Input Table: `users`

user_id	name	signup_date
1	Alice	2025-04-10
2	Bob	2025-03-20
3	Carlos	2025-04-29
4	Diana	2025-05-10
5	Emma	2025-04-01

📊 Output

user_id	name	signup_date
1	Alice	2025-04-10
3	Carlos	2025-04-29
4	Diana	2025-05-10

Assuming current date is 2025-05-15.

✅ Why It Matters

- Understand date filtering with `INTERVAL`
- Write time-based logic for dashboards and reports

> ⚠️ **Note:** SQL date functions vary by database. Check whether yours uses `NOW()`, `GETDATE()`, or `CURRENT_DATE`.

Query #3: Find the Most Popular Product

Skill: 📊 Aggregation + 🔽 Ranking

Scenario: You want to feature the most purchased product on your homepage.

💻 SQL

```sql
SELECT product_name, COUNT(*) AS purchase_count
FROM orders
GROUP BY product_name
ORDER BY purchase_count DESC
LIMIT 1;
```

🧾 Input Table: `orders`

order_id	product_name	user_id
101	Coffee	1
102	Tea	2
103	Coffee	3
104	Coffee	4
105	Tea	1
106	Smoothie	5

📊 Output

product_name	purchase_count
Coffee	3

✅ Why It Matters

- Learn how to rank values using `ORDER BY`
- Combine with `LIMIT` to extract top entries

> 💡 **Bonus:** Use `WITH TIES` if your DBMS supports it, to handle popularity ties fairly.

Query #4: Join Users with Their Orders

Skill: 🔗 Joining Tables

Scenario: You need a report showing each order along with the user's name and country.

💻 SQL

```sql
SELECT o.order_id, o.product_name, u.name, u.country
FROM orders o
INNER JOIN users u ON o.user_id = u.user_id;
```

🧾 Input Tables

`users`

user_id	name	country
1	Alice	USA
2	Bob	Canada
3	Carlos	USA
4	Diana	UK

`orders`

order_id	product_name	user_id
101	Coffee	1
102	Tea	2
103	Coffee	3
104	Smoothie	4

📊 Output

order_id	product_name	name	country
101	Coffee	Alice	USA
102	Tea	Bob	Canada
103	Coffee	Carlos	USA
104	Smoothie	Diana	UK

✅ Why It Matters

- Learn how to connect data from multiple tables
- Use aliases (`o` , `u`) to improve readability

> 🧠 **Note:** `INNER JOIN` only returns matches. Use `LEFT JOIN` to include unmatched rows too.

Query #5: Calculate Running Total of Orders

Skill: 🪟 Window Functions

Scenario: You want to show cumulative daily orders for a time-series dashboard.

💻 SQL

```sql
SELECT
  order_date,
  COUNT(*) AS daily_orders,
  SUM(COUNT(*)) OVER (ORDER BY order_date) AS running_total
FROM orders
GROUP BY order_date
ORDER BY order_date;
```

🧾 Input Table: `orders`

order_id	order_date	user_id
201	2025-05-01	1
202	2025-05-02	2
203	2025-05-03	3
204	2025-05-04	1
205	2025-05-05	2

📊 Output

order_date	daily_orders	running_total
2025-05-01	1	1
2025-05-02	1	2
2025-05-03	1	3
2025-05-04	1	4
2025-05-05	1	5

✅ Why It Matters

- Learn how to track **cumulative metrics** with `SUM OVER`
- Build time-based visualizations with SQL alone

> 💡 Always include `ORDER BY` inside `OVER()` to define accumulation flow.

Query #6: Rank Users by Total Spending

Skill: 🪟 Window Functions + 📊 Aggregation

Scenario: Marketing wants to reward the top 3 spenders.

💻 SQL

```sql
SELECT
  user_id,
  SUM(order_value) AS total_spent,
  RANK() OVER (ORDER BY SUM(order_value) DESC) AS spend_rank
FROM orders
GROUP BY user_id;
```

📄 Input Table: `orders`

order_id	user_id	order_value
301	1	25.00
302	2	40.00
303	1	30.00
304	3	100.00
305	2	20.00

📊 Output

user_id	total_spent	spend_rank
3	100.00	1
2	60.00	2
1	55.00	3

✅ Why It Matters

- Combine `GROUP BY` with ranking functions
- Use `RANK()` for leaderboards and business performance reports

> 💡 Use `DENSE_RANK()` if you want no gaps in tied ranks.

Query #7: Find First Purchase Date for Each User

Skill: 📊 Aggregation

Scenario: You want to track user acquisition cohorts by their first purchase.

💻 SQL

```sql
SELECT
  user_id,
  MIN(order_date) AS first_purchase_date
FROM orders
GROUP BY user_id;
```

🧾 Input Table: `orders`

order_id	user_id	order_date
401	1	2025-05-01
402	2	2025-05-03
403	1	2025-05-05
404	3	2025-05-04
405	2	2025-05-06

📊 Output

user_id	first_purchase_date
1	2025-05-01
2	2025-05-03
3	2025-05-04

✅ Why It Matters

- Find the **earliest event** in a user's lifecycle
- Useful for cohort analysis, segmentation, and funnels

> 💡 You can join this back to the original table to get full row details for each user's first event.

Query #8: Get First Order Row for Each User

Skill: 🪟 Window Function + 🧩 CTE

Scenario: You want to retrieve full details of each user's first purchase, not just the date.

💻 SQL

```sql
WITH ranked_orders AS (
  SELECT *,
         ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY order_date) AS rn
  FROM orders
)
SELECT order_id, user_id, order_date, order_value
FROM ranked_orders
WHERE rn = 1;
```

📄 Input Table: `orders`

order_id	user_id	order_date	order_value
501	1	2025-05-01	25.00
502	2	2025-05-03	40.00
503	1	2025-05-05	30.00
504	3	2025-05-04	100.00
505	2	2025-05-06	20.00

📊 Output

order_id	user_id	order_date	order_value
501	1	2025-05-01	25.00
502	2	2025-05-03	40.00
504	3	2025-05-04	100.00

✅ Why It Matters

- Use `ROW_NUMBER()` to isolate the first row per group
- Combines window functions with CTEs for flexible logic

> ⚠️ Always define `ORDER BY` inside the window function to control how rows are ranked.

Query #9: Find Products Purchased More Than Once

Skill: 📊 Aggregation + 🔽 Filtering After Grouping

Scenario: You are analyzing product performance and want to find those with repeat sales.

💻 SQL

```sql
SELECT product_name, COUNT(*) AS order_count
FROM orders
GROUP BY product_name
HAVING COUNT(*) > 1;
```

📄 Input Table: `orders`

order_id	product_name
601	Coffee
602	Tea
603	Coffee
604	Smoothie
605	Tea

📊 Output

product_name	order_count
Coffee	2
Tea	2

✅ Why It Matters

- Learn how to **filter grouped results** using `HAVING`
- Detect repeat purchases or frequent behaviors

> 🧠 `HAVING` filters **after** aggregation, while `WHERE` filters **before**.

Query #10: Detect Duplicate User Emails

Skill: 🔽 Data Quality Check + 📊 Aggregation

Scenario: You are auditing your user table to find duplicate email addresses.

💻 SQL

```sql
SELECT email, COUNT(*) AS occurrences
FROM users
GROUP BY email
HAVING COUNT(*) > 1;
```

📄 Input Table: `users`

user_id	email
1	alice@example.com
2	bob@example.com
3	carlos@example.com
4	alice@example.com
5	diana@example.com

📊 Output

email	occurrences
alice@example.com	2

✅ Why It Matters

- Perform essential **data validation** with SQL
- Spot duplicates and ensure database integrity

> 💡 Combine with a `JOIN` to fetch full details of duplicate rows.

Recap: What You Just Learned

10 Practical SQL Queries for Data Scientists

Query	Use Case	Key Concepts	Skills
Count Users by Country	Aggregation summary	`GROUP BY`, `COUNT`	📊
Recent Signups	Time-based filtering	`WHERE`, `INTERVAL`	🔽
Most Popular Product	Top-N ranking	`ORDER BY`, `LIMIT`	📊 🔽
Join Orders with Users	Combine tables	`JOIN`, aliases	🔗
Running Total of Orders	Cumulative metrics	`SUM() OVER`, `GROUP BY`	🪟
Rank Top Spenders	Leaderboard logic	`RANK() OVER`, `SUM()`	🪟 📊
First Purchase Date	Cohort tracking	`MIN()`, `GROUP BY`	📊
First Order Row per User	Full row extraction	`ROW_NUMBER()`, `CTE`	🪟 🧩
Products Bought More Than Once	Frequency analysis	`HAVING`, `COUNT(*)`	📊 🔽
Duplicate Emails	Data quality validation	`GROUP BY`, `HAVING`	🔽 📊

📘 Skill Legend

- 📊 Aggregation
- 🔽 Filtering
- 🔗 Join
- 🪟 Window Function
- 🧩 CTE (Common Table Expression)

> Keep practicing these patterns to build confidence and fluency.

Enjoyed this guide?

Let’s keep the conversation going!

👍 **Like** if you found it helpful

💬 **Comment** with your thoughts or questions

🔁 **Repost** to share with your network

🔖 **Save** for future reference

➕ **Follow** for more practical content like this

Thank you for exploring this guide with me.

Let us continue the journey.

Mostapha Kalami Heris, PhD

Applied AI and Machine Learning Scientist

`linkedin.com/in/smkalami`